Logical Methods

in

Automated Hardware and Software Verification

Georg Weissenbacher
www.georg.weissenbacher.name

Education:

Sept. 2010	**Doctor of Philosophy** University of Oxford
2003	**Diplom-Ingenieur** TU Graz (Telematik)

Research positions: (before TU Wien)

2010 to 2012	**Princeton University** Postdoctoral Research Associate
2005 to 2010	**ETH Zürich** Research Assistant
2003 & 2008	**Microsoft Research** Summer Intern
2004 to 2005	**Austrian Institute of Technology** Software Engineer

Teaching Experience (before TU Wien)

2011 **Lecturer, Princeton University**

- Automated Verification & Software Model Checking

2005 to 2010 **Teaching Assistant, ETHZ**

- Digitaltechnik
- Formal Verification

Digitaltechnik

Teaching Experience

Semantik v. Programmier-sprachen
(w. F. Zuleger)

Introduction to Logical Methods in Comp. Sci.
(w. LogiCS Faculty)

Seminar Formal Methods

Computer Aided Verification
(w. Igor Konnov)

Software Model Checking

Formale Methoden d. Informatik
(w. Zuleger, . . .)

master

- -

bachelor

Programm- und Systemverifikation
(w. Josef Widder)

Teaching Experience

Semantik v. Programmier-sprachen
(w. F. Zuleger)

Introduction to Logical Methods in Comp. Sci.
(w. LogiCS Faculty)

Seminar Formal Methods

Computer Aided Verification
(w. Igor Konnov)

Software Model Checking

Formale Methoden d. Informatik
(w. Zuleger, . . .)

master

bachelor

Programm- und Systemverifikation
(w. Josef Widder)

Doctoral College Board Member

184.741 Programm- und Systemverifikation

(comments from 2013-2015; 90 bachelor students)

Die Vorträge von Prof. Weissenbacher waren **großartig**. Großes Kompliment an Sie. **Ich habe bisher keinen so angenehmen Vortragenden erlebt**. Es war immer spannend und interessant.

sehr gute Folien und toller Vortragsstil (besonders Georg Weißenbacher)

Ich hätte mir im Vorhinein nicht gedacht, dass es so interessant wird, aber ich war **sehr positiv überrascht**.

die netten und kompetenten Vorträge der Vortragenden Georg Weissenbacher und Josef Widder; der makellose englische Akzent des Vortragenden Georg Weissenbacher (wahrlich eine Wohltat für die Ohren)

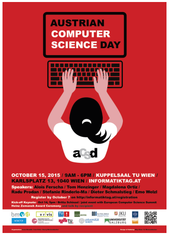
. . . war die Lehrveranstaltung, ihre Organisation betreffend, wirklich vorbildhaft. Vor allem die Erreichbarkeit des Lehrveranstaltungsteams (TISS-Forum) war **überdurchschnittlich gut**.

Funding & Projects

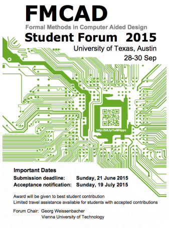
2011	**Vienna Research Group for Young Investigators**
	“Heisenbugs: From Detection to Explanation”
	WWTF Funding: EUR 1.5m
2014	**FWF Doctoral College**
	“Logical Methods in Computer Science”
	Co-author of proposal, board member
	FWF Overall Funding: 2.8m (15 PhD positions)
2015	**RiSE Research Network**
	Principal Investigator
	FWF Overall Funding: 3.6m, FORSYTE share: 625k
2016	**Microsoft European PhD Scholarship**
	Funding: 110k

Academic Service

Event organization:


AUSTRIAN COMPUTER SCIENCE DAY
OCTOBER 15, 2015 / 9AM - 6PM / KUPPELSAAL TU WIEN / KARLSPLATZ 13, 1040 WIEN / INFORMATIKTAG.AT
Speakers: Alois Ferscha / Tom Henzinger / Magdalena Ortiz / Radu Prodan / Stefanie Rinderle-Ma / Dieter Schmalstieg / Emo Welzl
Register by October 7 on http://informatiktag.at/registration

Informatiktag'15


FMCAD
Formal Methods in Computer Aided Design
Student Forum 2015
University of Texas, Austin
28-30 Sep
Important Dates
Submission deadline: Sunday, 21 June 2015
Acceptance notification: Sunday, 19 July 2015
Award will be given to best student contribution
Limited travel assistance available for students with accepted contributions
Forum Chair: Georg Weissenbacher
Vienna University of Technology

FMCAD Student Forum '15

SAT/SMT Summer School
July 10-12, 2014
Semmering, Austria
4th International Summer School on Satisfiability/Satisfiability Modulo Theories
Vienna Summer of Logic 2014
Important Dates
Speakers
Sponsored by
Research

SAT/SMT Summer School '14

LOVE'16

spring school

Interpolation Workshop '13-15

PC membership:

- Conference co-chair: FMCAD '17 (TU Wien), CAV '18
- Conference PC: CAV '13-'15; ICCAD '15-'16; FMCAD '13-'15;
- Workshop PC: DUHDe '15; CREST '15; SMT '14; SV-COMP '12, . . .

What happened since I arrived at TU Wien...

Toyota Prius

(New York Times, Feb. 12, 2014)

Toyota Motor is recalling all of the 1.9 million newest-generation Prius vehicles it has sold worldwide because of a programming error . . .

What happened since I arrived at TU Wien...

Toyota Prius

(New York Times, Feb. 12, 2014)

Toyota Motor is recalling all of the 1.9 million newest-generation Prius vehicles it has sold world-wide because of a programming error ...

Heathrow Airport

(The Guardian, December 2014)

An unprecedented systems failure was responsible for the air traffic control chaos [...] "In this instance a transition between the two states caused a failure in the system which has not been seen before," ...

Heathrow
Making every journey better

Lufthansa Airbus A321

(Spiegel, March 20, 2015)

Beinahe wäre ein Airbus A321 der Lufthansa mit 109 Passagieren auf dem Flug von Bilbao nach München abgestürzt – irregeleitete Bordcomputer hatten die Kontrolle übernommen.

Lufthansa Airbus A321

(Spiegel, March 20, 2015)

Beinahe wäre ein Airbus A321 der Lufthansa mit 109 Passagieren auf dem Flug von Bilbao nach München abgestürzt – irregeleitete Bordcomputer hatten die Kontrolle übernommen.

Boeing 787 Dreamliner

(The Guardian, May 2015)

The US air safety authority has issued a warning and maintenance order over a software bug that causes a complete electric shutdown of Boeing's 787 . . .

Heartbleed Bug

(CNN, April 9, 2014)

A major online security vulnerability dubbed “Heartbleed” could put your personal information at risk, including passwords, credit card information and e-mails.

Heartbleed Bug

(CNN, April 9, 2014)

A major online security vulnerability dubbed “Heartbleed” could put your personal information at risk, including passwords, credit card information and e-mails.

Resolved

in 184.741 (P&SV)

Heartbleed Bug

(CNN, April 9, 2014)

A major online security vulnerability dubbed “Heartbleed” could put your personal information at risk, including passwords, credit card information and e-mails.

Resolved **in 184.741 (P&SV)**

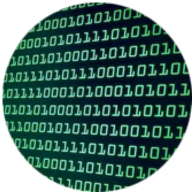
Rowhammer Bug

(InfoWorld, March 9, 2015)

. . . with certain varieties of DRAM an attacker can create privilege escalations by simply repeatedly accessing a given row of memory.

Software and integrated circuits are everywhere

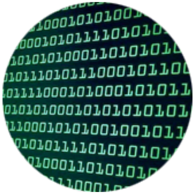
Software and integrated circuits are everywhere



10^6 lines of code

70 micro-processors

Huge Effort Spent on V&V



Software verification
50% of development time
[Myers 1979–2012]

Hardware validation
35% of development time
[Abramovici 2006]

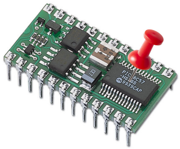
Establishing correctness

Establishing correctness

Finding bugs

Establishing correctness

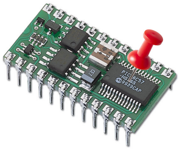
Finding bugs



Locating faults

Establishing correctness

Finding bugs

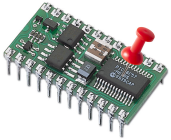


Locating faults

$$\left.\begin{array}{l}\text{Establishing correctness}\\\text{Finding bugs}\\\text{Locating faults}\end{array}\right\}\ \text{Automated Verification}$$

Scalable Software
Model Checking
[CAV'14]

Efficient Detection
of "Deep" Bugs
[FMSD'15] (CAV'13),
[FM'15]



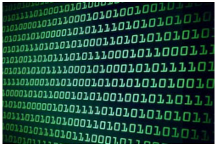
Fault Localization
in Post-Silicon
[ICCAD'14]

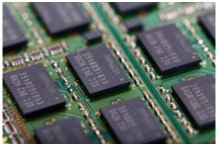
My Habilitation

Logical foundations
[JAR'16] (single auth. SAT'12)

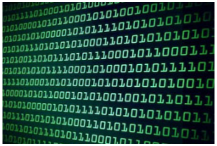
State-of-the-Art
[Proc. IEEE'15]

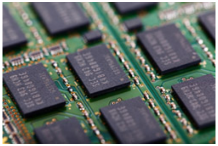
Model Checking 101





Logic





T

(transitions)

T

$$\begin{tikzcd} s \arrow[r, bend left, "\mathbf{T}"] & s' \end{tikzcd}$$

T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

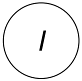
T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

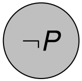
$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

The **Model Checking** problem:



“starting states”



“bad states”

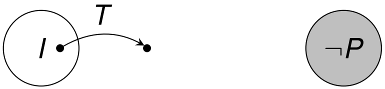
T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

The **Model Checking** problem:



“starting states” “bad states”

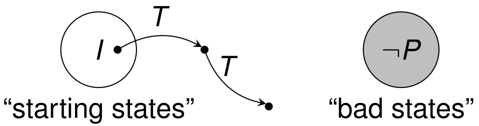
T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

The **Model Checking** problem:



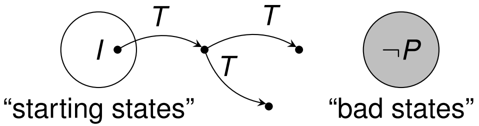
T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

The **Model Checking** problem:



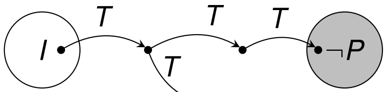
T

$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

The **Model Checking** problem:



"starting states"

"bad states"

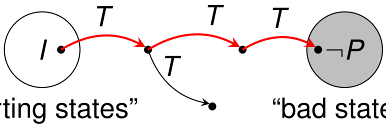
T

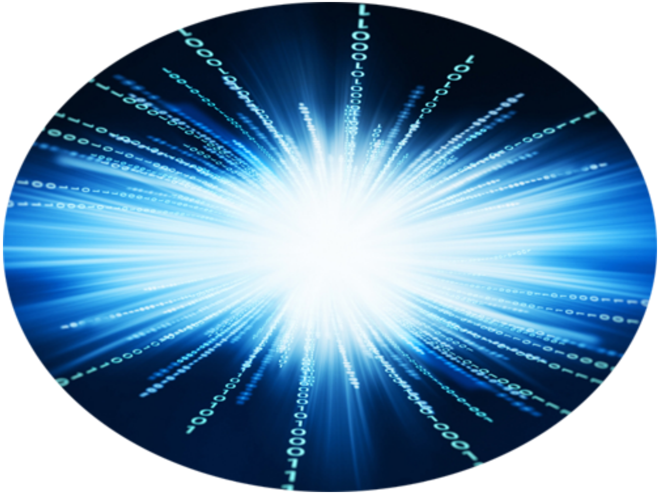
$$\begin{tikzcd} s \arrow[r, bend left] & s' \end{tikzcd}$$

$$\begin{tikzcd} \langle pc \mapsto 2, x \mapsto 1 \rangle \arrow[r, bend left, "T"] & \langle pc \mapsto 3, x \mapsto 2 \rangle \end{tikzcd}$$

(T: operational semantics of program or circuit)

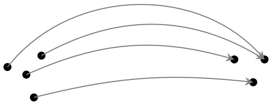
The **Model Checking** problem:



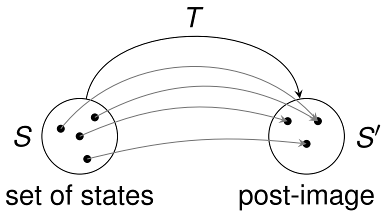


State Space Explosion

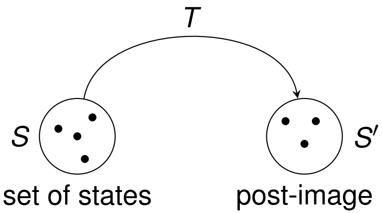
Why explore states one by one?



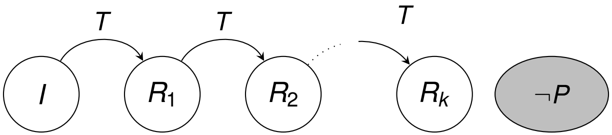
Why explore states one by one?



Why explore states one by one?



$$S' \quad = \quad T(S) \overset{\text{def}}{=} \{s' \mid T(s, s') \wedge s \in S\}$$


T
T
T
I
R_1
R_2
R_k
$\neg P$

How do we efficiently represent sets of states?

Logical Formulas!

$$\underbrace{V}_{\text{program variables, registers, latches, signals, \ldots}}$$

How do we efficiently represent sets of states?

Logical Formulas!

$$F(\underbrace{V}_{\substack{\text{program variables,}\\ \text{registers, latches,}\\ \text{signals, \ldots}}})$$

How do we efficiently represent sets of states?

Logical Formulas!

$(x > 0)$ represents $\{s \mid s(x) > 0\}$

And what about transitions?

Binary Relations!

$$T(\, V, \underbrace{V'}_{\text{target states}}\,)$$

And what about transitions?

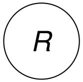
Binary Relations!

$(x' = x + 1)$ represents $\{\langle s, s' \rangle \mid s'(x) = s(x) + 1\}$

And what about transitions?

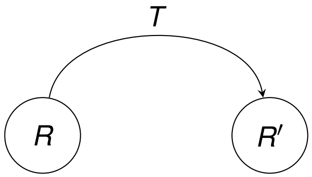
Binary Relations!

$$\underbrace{(x' = x + 1)}_{\texttt{x++}} \quad \text{represents} \quad \{\langle s, s' \rangle \mid s'(x) = s(x) + 1\}$$



R

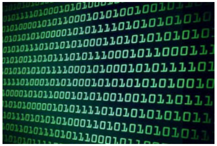
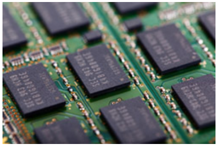
$$\begin{tikzcd} R \arrow[rr, bend left, "T"] & & R' \end{tikzcd}$$



$$R'(V') \quad \stackrel{\text{def}}{=} \quad \exists V \,. \quad R(V) \quad \wedge \quad T(V, V')$$

$$\begin{tikzcd}R \arrow[r, bend left, "T"] & R' \arrow[l, bend left, "T^{-1}"]\end{tikzcd}$$

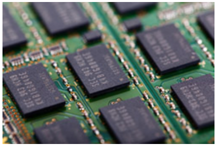
$$\begin{aligned}
R'(V') &\overset{\text{def}}{=} \exists V\,.\ R(V) \land T(V, V') \\
R(V) &\overset{\text{def}}{=} \exists V'\,.\ T(V, V') \land R'(V')
\end{aligned}$$

$$\underbrace{\hspace{6cm}}_{\mathbf{T}}$$

(transition relation)

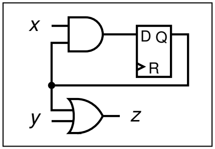
```
if (x>0) {
  x = x - 1;
} else {
  x = x + 1;
}
```



T

(transition relation)

```
if (x>0) {
  x = x - 1;
} else {
  x = x + 1;
}
```



T

(transition relation)

$$\underbrace{\boxed{\begin{array}{ll}\text{1:} & \texttt{if (x>0)} \\ \text{2:} & \texttt{\ \ x = x - 1;} \\ \text{3:} & \texttt{else} \\ \text{4:} & \texttt{\ \ x = x + 1;} \\ \text{5:} & \texttt{assert (x}\geq\texttt{0);}\end{array}}}_{T(\langle pc, x\rangle, \langle pc', x'\rangle)}$$

$$\wedge\left(\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\right)$$

```
1:  if (x>0)
2:    x = x - 1;
3:  else
4:    x = x + 1;
5:  assert (x≥0);
```

$$T(\langle pc, x\rangle, \langle pc', x'\rangle) \stackrel{\text{def}}{=}$$

$$\bigwedge \left(\begin{array}{l} (pc = 1) \;\wedge\; (x > 0) \;\Rightarrow\; (pc' = 2) \;\wedge\; (x' = x) \\ \\ \\ \end{array} \right)$$

```
1:  if (x>0)
2:    x = x - 1;
3:  else
4:    x = x + 1;
5:  assert (x≥0);
```

$$T(\langle pc, x\rangle, \langle pc', x'\rangle) \overset{\text{def}}{=}$$

$$\bigwedge \left(\begin{array}{lllllll} (pc = 1) & \wedge & (x > 0) & \Rightarrow & (pc' = 2) & \wedge & (x' = x) \\ (pc = 1) & \wedge & \neg(x > 0) & \Rightarrow & (pc' = 4) & \wedge & (x' = x) \\ & & & & & & \end{array} \right)$$

```
1:  if (x>0)
2:     x = x - 1;
3:  else
4:     x = x + 1;
5:  assert (x≥0);
```

$$T(\langle pc, x\rangle, \langle pc', x'\rangle) \overset{\text{def}}{=}$$

$$\bigwedge \left(\begin{array}{llllll} (pc = 1) & \wedge & (x > 0) & \Rightarrow & (pc' = 2) & \wedge & (x' = x) \\ (pc = 1) & \wedge & \neg(x > 0) & \Rightarrow & (pc' = 4) & \wedge & (x' = x) \\ (pc = 2) & & & \Rightarrow & (pc' = 5) & \wedge & (x' = x - 1) \end{array} \right)$$

```
1:  if (x>0)
2:    x = x - 1;
3:  else
4:    x = x + 1;
5:  assert (x≥0);
```

$$T(\langle pc, x\rangle, \langle pc', x'\rangle) \overset{\text{def}}{=}$$

$$\bigwedge \left(\begin{array}{lllllll} (pc = 1) & \wedge & (x > 0) & \Rightarrow & (pc' = 2) & \wedge & (x' = x) \\ (pc = 1) & \wedge & \neg(x > 0) & \Rightarrow & (pc' = 4) & \wedge & (x' = x) \\ (pc = 2) & & & \Rightarrow & (pc' = 5) & \wedge & (x' = x - 1) \\ (pc = 4) & & & \Rightarrow & (pc' = 5) & \wedge & (x' = x + 1) \end{array} \right)$$

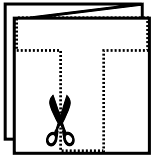
```
1:  if (x>0)
2:    x = x - 1;
3:  else
4:    x = x + 1;
5:  assert (x≥ 0);
```

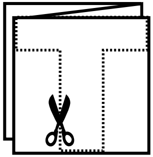
$$\underbrace{}_{T(\langle pc, x\rangle, \langle pc', x'\rangle) \overset{\text{def}}{=}}$$

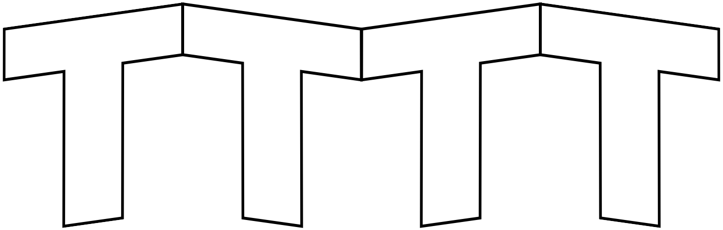
$$\bigwedge \left(\begin{array}{lllllll} (pc = 1) & \wedge & (x > 0) & \Rightarrow & (pc' = 2) & \wedge & (x' = x) \\ (pc = 1) & \wedge & \neg(x > 0) & \Rightarrow & (pc' = 4) & \wedge & (x' = x) \\ (pc = 2) & & & \Rightarrow & (pc' = 5) & \wedge & (x' = x - 1) \\ (pc = 4) & & & \Rightarrow & (pc' = 5) & \wedge & (x' = x + 1) \end{array} \right)$$

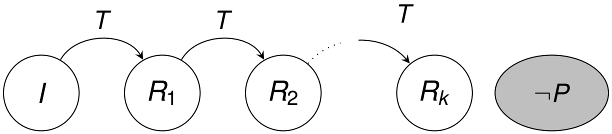
$$P(V) \overset{\text{def}}{=} (pc = 5) \Rightarrow (x \geq 0)$$

$$I(V) \overset{\text{def}}{=} (pc = 1)$$





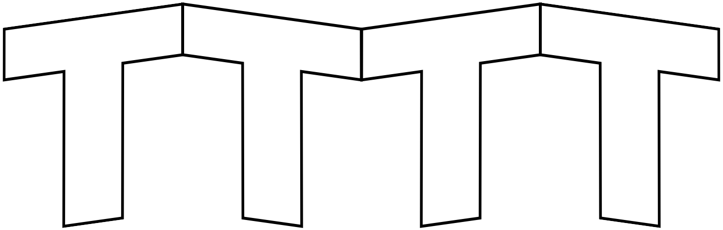


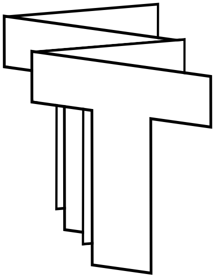


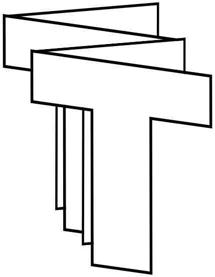
$$I(V_0) \wedge \left(\bigwedge_{i=1}^{k} T(V_{i-1}, V_i) \right) \wedge \neg P(V_k)$$

“Can property *P* be violated in *k* steps?”

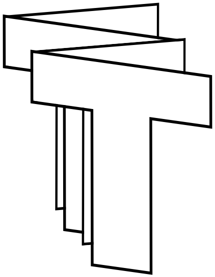
(here, property = assertion over variables)







$T^{\langle 4 \rangle}$



$T^{\langle n \rangle}$

$$\texttt{i}' = \texttt{i + 1}$$

$\mathtt{i}' = \mathtt{i} + n$

$$\exists n \in \mathbb{N}\,.\ \texttt{i}' = \texttt{i} + n$$

$\exists n \in \mathbb{N} .\ \mathtt{i}' = \mathtt{i} + n$

$$\exists n \in \mathbb{N}.\ \mathtt{i}' = \mathtt{i} + n$$

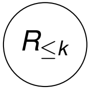
- $T^{\langle n \rangle}$ is *accelerated* version of T:

$$\begin{tikzcd} I \arrow[rr, bend left, "\exists n . T^{\langle n \rangle}"] & & \neg P \end{tikzcd}$$

- computable if $T^{\langle n \rangle}$ is Presburger-definable (for instance)
    - but not computable in general

$$\begin{tikzcd}
I \arrow[r, bend left, "T"] & R_1 \arrow[r, bend left, "T"] & R_2 \arrow[r, dotted, no head] & \cdots \arrow[r, bend left, "T"] & R_k
\end{tikzcd}$$

$$\begin{tikzcd} I \arrow[r, bend left, "T"] & R_1 \arrow[r, bend left, "T"] & R_2 \arrow[r, bend left, dashed, "T"] & R_k \end{tikzcd}$$

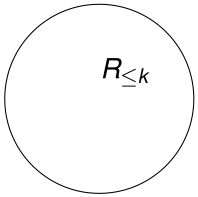


$$R_{\leq k} = \bigcup_{i=0}^{k} R_i \quad (\text{with } R_0 \overset{\text{def}}{=} I)$$

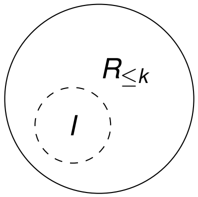
$$\begin{tikzcd} R_{\leq k} \arrow[loop above, "T"] \end{tikzcd}$$

$$R_{\leq k} = \bigcup_{i=0}^{k} R_i \quad (\text{with } R_0 \overset{\text{def}}{=} I)$$

- "Fixed point" if T cannot escape $R_{\leq k}$

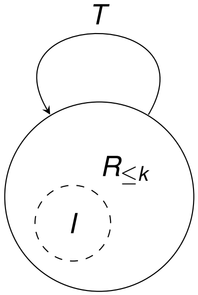


System is safe if:



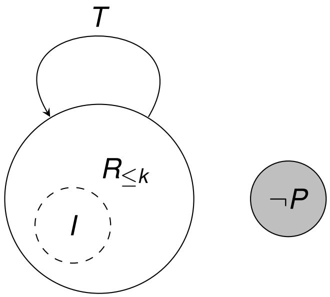
System is safe if:

- $R_{\leq k}$ contains I



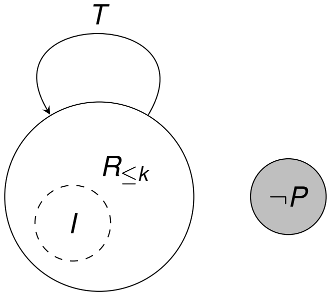
System is safe if:

- $R_{\leq k}$ contains I
- T cannot leave $R_{\leq k}$



System is safe if:

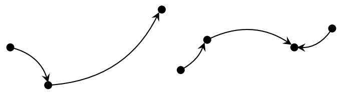
- $R_{\leq k}$ contains I
- T cannot leave $R_{\leq k}$
- $R_{\leq k}$ does not overlap with $\neg P$

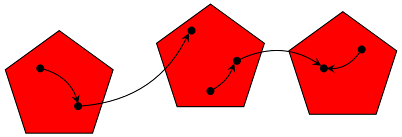


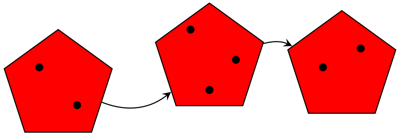
System is safe if:

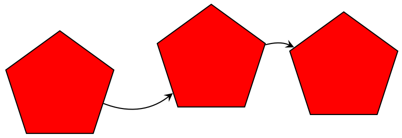
- $R_{\leq k}$ contains I
- T cannot leave $R_{\leq k}$
- $R_{\leq k}$ does not overlap with $\neg P$

$R_{\leq k}$ challenging to find for *concrete industrial-size* systems



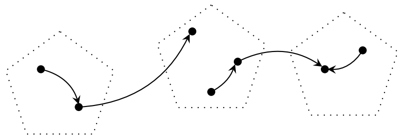


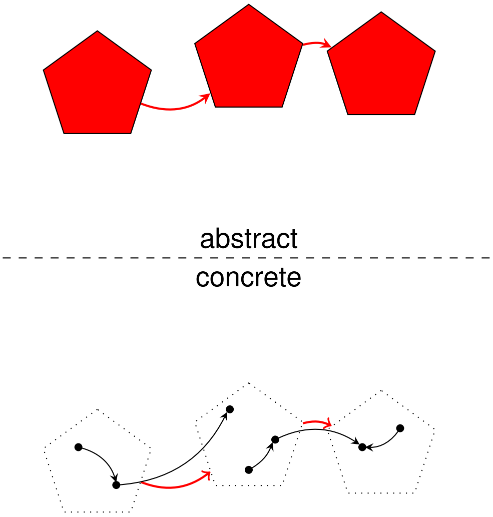


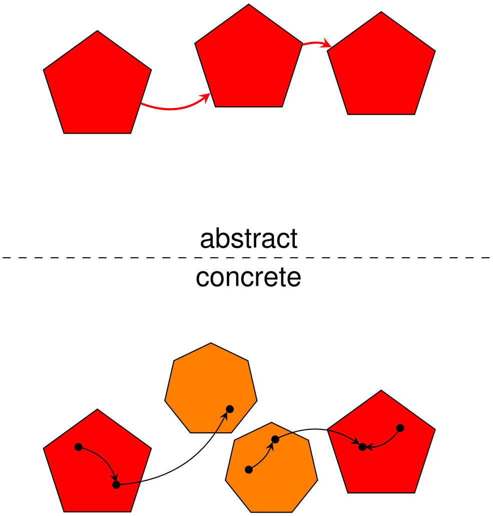


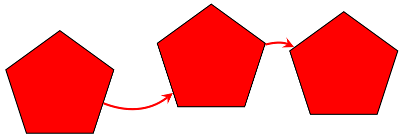
abstract

concrete



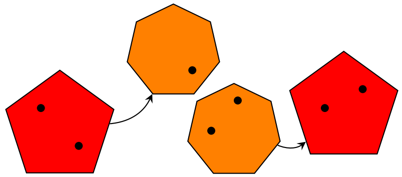

abstract
concrete

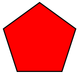

abstract
concrete

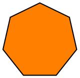


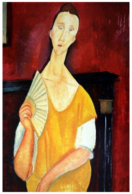
abstract

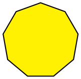
less abstract

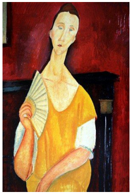


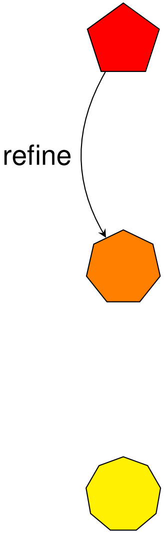


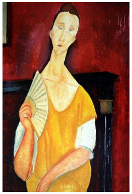


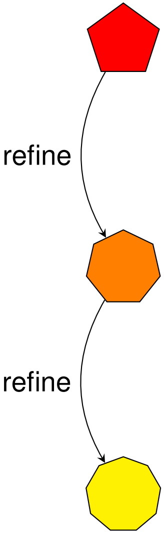




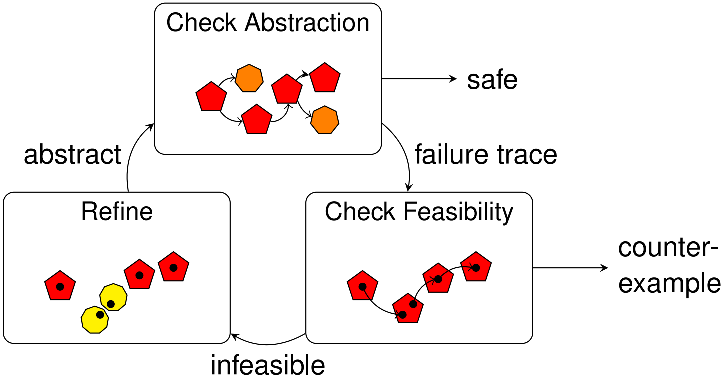



refine

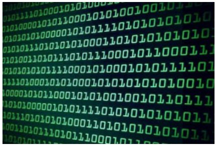


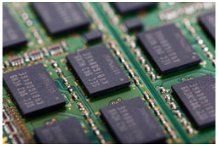

refine
refine

Counterexample-guided Abstraction Refinement (CEGAR)



Model Checking in Practice





$\underbrace{\qquad\qquad\qquad\qquad}_{\textbf{T}}$

$\downarrow$

Satisfiability Solver

(like linear programming, but for first-order/propositional logic)

Satisfiability Solvers

Z3 Yices2

MathSAT 5

PicoSAT Boolector Lingeling

- Satisfiability of First-Order/Propositional Logic
- Solve large instances with *hundreds of thousands of variables*
- Cornerstone of modern-day formal verification

Automated Verification in Industry

Software

Microsoft Research

Yogi Pex

SLAM SAGE

facebook

Google

Hardware

IBM

Sixth Sense

SYNOPSYS®

cadence®

What we want to verify:

What we want to verify:

What we can verify:

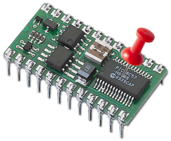
What we want to verify:

What we can verify:

My research: **Push the Boundary**

Scalable Software Model Checking
[CAV'14]

Efficient Detection of "Deep" Bugs
[FMSD'15] (CAV'13), [FM'15]



Fault Localization in Post-Silicon
[ICCAD'14]

My Habilitation

Logical foundations
[JAR'16] (single auth. SAT'12)

State-of-the-Art
[Proc. IEEE'15]

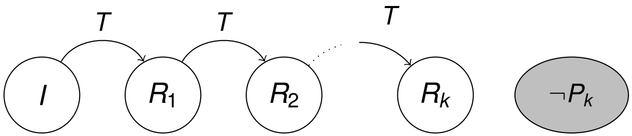
Logical foundations

State-of-the-Art

Schlaipfer, Weissenbacher:
Labelled Interpolation Systems for Hyper-Resolution, Clausal, and Local Proofs.
Journal of Automated Reasoning ’16

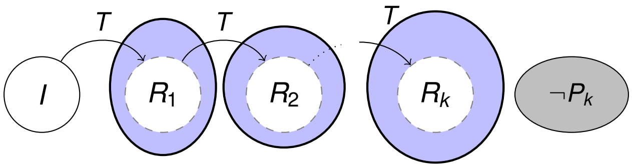
Vizel, Weissenbacher, Malik:
Boolean Satisfiability Solvers and Their Applications in Model Checking.
Proceedings of the IEEE ’15

Interpolation-based Hardware Model Checking [Proc. IEEE'15]



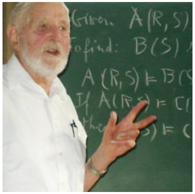
- Exact reachability retards convergence

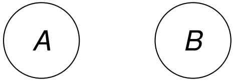
Interpolation-based Hardware Model Checking [Proc. IEEE'15]



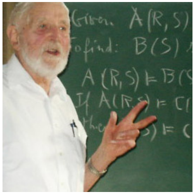
- Exact reachability retards convergence
- Over-approximate R_i instead?

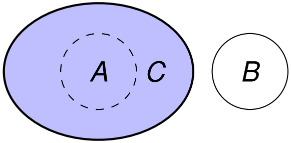
Craig's Interpolation Theorem





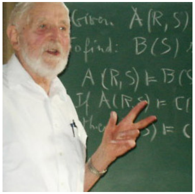
Craig’s Interpolation Theorem

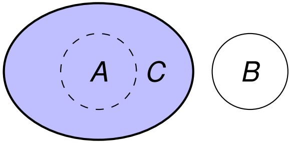




C “simpler” than *A*

Craig's Interpolation Theorem

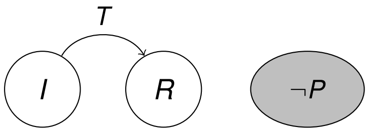




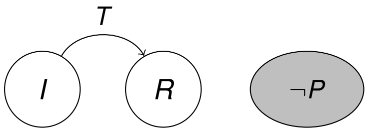
C "simpler" than A

$$\text{if } (A(V, V') \wedge B(V', V'') \models \bot)$$
$$\Downarrow$$
$$\exists C(V')$$
$$\text{s.t.}$$
$$A(V, V') \models C(V')$$
$$B(V', V'') \models \neg C(V')$$

Interpolation-based Hardware Model Checking [Proc. IEEE'15]

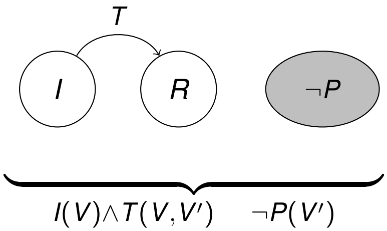


Interpolation-based Hardware Model Checking [Proc. IEEE'15]



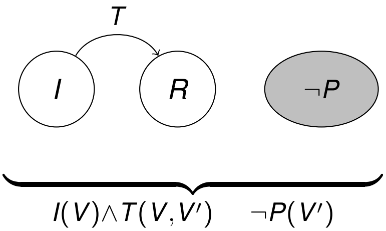
$$\underbrace{I(V) \wedge T(V,V') \quad \neg P(V')}$$

Interpolation-based Hardware Model Checking [Proc. IEEE'15]



$$\underbrace{I(V) \wedge T(V, V')}_{A(V,V')} \qquad \underbrace{\neg P(V')}_{B(V')}$$

Interpolation-based Hardware Model Checking [Proc. IEEE'15]

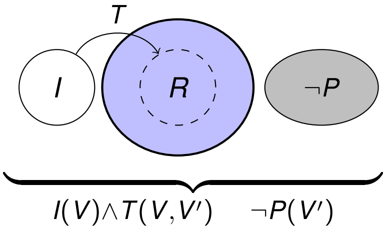


$$\underbrace{I(V) \wedge T(V, V')}_{A(V,V')} \qquad \underbrace{\neg P(V')}_{B(V')}$$

$$\downarrow$$

$$C(V')$$

Interpolation-based Hardware Model Checking [Proc. IEEE'15]



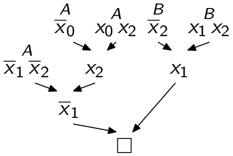
$$\underbrace{I(V) \wedge T(V, V')}_{A(V,V')} \qquad \underbrace{\neg P(V')}_{B(V')}$$

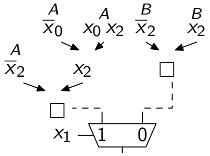
$$\downarrow$$

$$C(V')$$

Generalized Interpolation [Journal of Automated Reasoning'16]

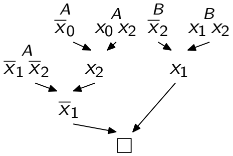
- Interpolants from Propositional/First-Order Refutation Proofs

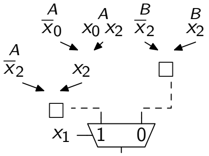




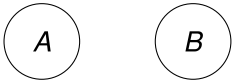
Generalized Interpolation [Journal of Automated Reasoning'16]

- Interpolants from Propositional/First-Order Refutation Proofs



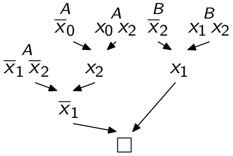


- Systematic variation of logical strength and structure

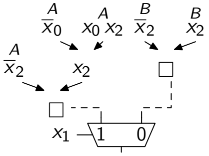


Generalized Interpolation [Journal of Automated Reasoning'16]

- Interpolants from Propositional/First-Order Refutation Proofs

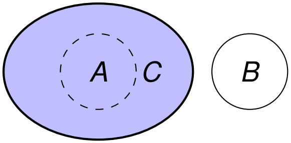


$\overset{A}{\overline{x}_0}$
$\overset{A}{x_0\, x_2}$
$\overset{B}{\overline{x}_2}$
$\overset{B}{x_1\, x_2}$
$\overset{A}{\overline{x}_1\, \overline{x}_2}$
x_2
x_1
$\overline{x}_1$
$\square$



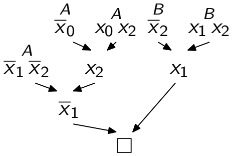
$\overset{A}{\overline{x}_0}$
$\overset{A}{x_0\, x_2}$
$\overset{B}{\overline{x}_2}$
$\overset{B}{x_2}$
$\overset{A}{\overline{x}_2}$
x_2
$\square$
$\square$
x_1
1
0

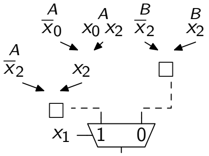
- Systematic variation of logical strength and structure



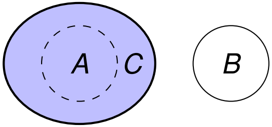
Generalized Interpolation [Journal of Automated Reasoning’16]

- Interpolants from Propositional/First-Order Refutation Proofs



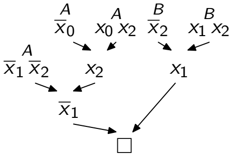


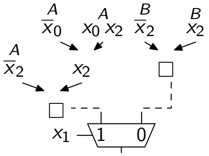
- Systematic variation of logical strength and structure



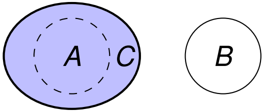
Generalized Interpolation [Journal of Automated Reasoning'16]

- Interpolants from Propositional/First-Order Refutation Proofs



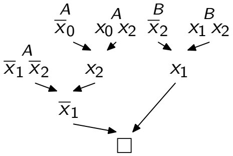


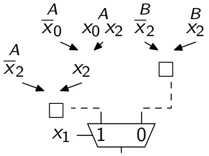
- Systematic variation of logical strength and structure



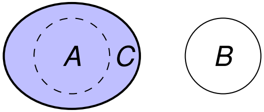
Generalized Interpolation [Journal of Automated Reasoning'16]

- Interpolants from Propositional/First-Order Refutation Proofs





- Systematic variation of logical strength and structure



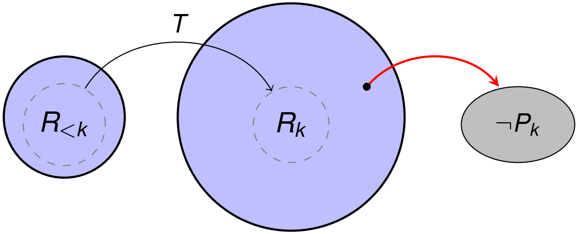
- Most general (propositional) interpolation algorithm to date

Scalable Software
Model Checking

Birgmeier, Bradley, Weissenbacher:
Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR).
Conference on Computer Aided Verification (CAV), 2014

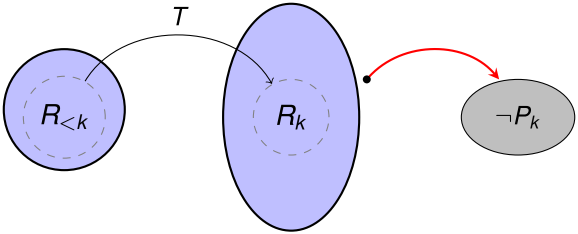
- Based on IC3, the leading *hardware* model checking algorithm
- state space in software is much larger or ∞
  - therefore, we need *abstraction*

Abstraction/Refinement for IC3 [Computer Aided Verification'14]



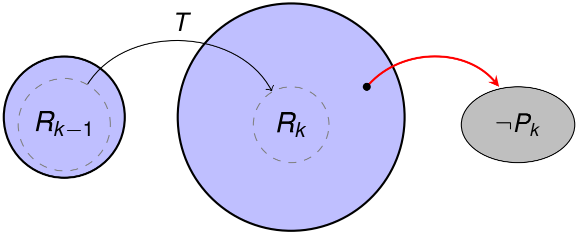
- IC3 refines approximations by eliminating unreachable states

Abstraction/Refinement for IC3 [Computer Aided Verification'14]

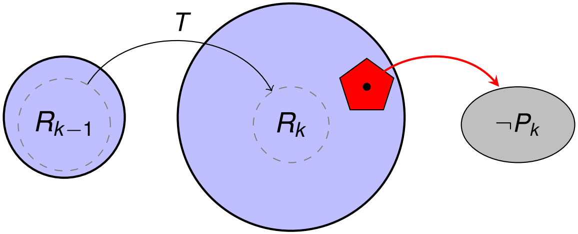


- IC3 refines approximations by eliminating unreachable states
  - in software, concrete-state refinement strategy not efficient

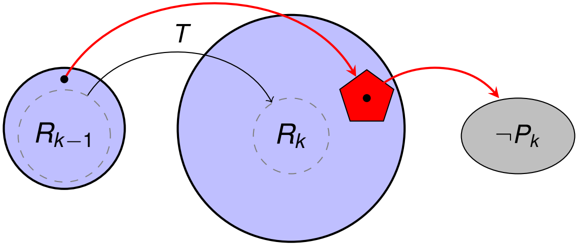
Abstraction/Refinement for IC3 [Computer Aided Verification'14]



Abstraction/Refinement for IC3 [Computer Aided Verification'14]

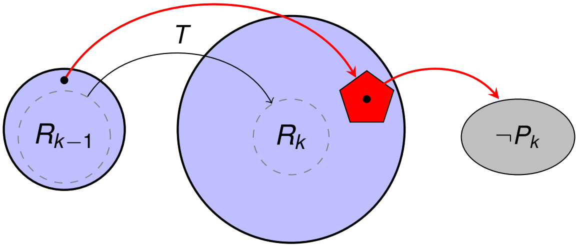


Abstraction/Refinement for IC3 [Computer Aided Verification'14]



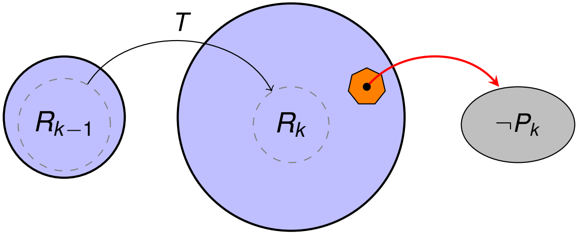
- Abstraction may introduce new predecessor
  - thwarts proof that bad state is unreachable

Abstraction/Refinement for IC3 [Computer Aided Verification'14]



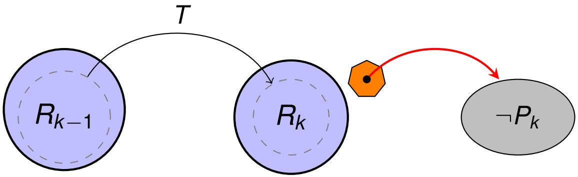
- Abstraction may introduce new predecessor
  - thwarts proof that bad state is unreachable
- CEGAR refinement requires *full* counterexample trace
  - in IC3, only single step available!

Abstraction/Refinement for IC3 [Computer Aided Verification'14]



- Abstraction may introduce new predecessor
  - thwarts proof that bad state is unreachable
- CEGAR refinement requires *full* counterexample trace
  - in IC3, only single step available!
- Our approach combines CEGAR and IC3
  - single-step refinement based on interpolation

Abstraction/Refinement for IC3 [Computer Aided Verification'14]



- Abstraction may introduce new predecessor
  - thwarts proof that bad state is unreachable
- CEGAR refinement requires *full* counterexample trace
  - in IC3, only single step available!
- Our approach combines CEGAR and IC3
  - single-step refinement based on interpolation

Abstraction/Refinement for IC3 [Computer Aided Verification'14]

- Our *prototype* tool successfully verifies more programs than winner of the 2014 Software Verification Competition
- New implementation for parallel software competed in

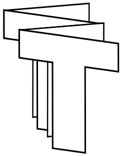
  Software Verification Competition '16

  - 4th in parallel software category
  - first 3 tools do bug-finding exclusively

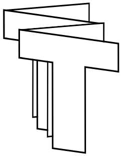
Efficient Detection
of “Deep” Bugs

Daniel Kroening, Matt Lewis, Georg Weissenbacher:
Under-approximating Loops in C Programs for Fast Counterexample Detection.
Journal for Formal Methods in Systems Design ’15

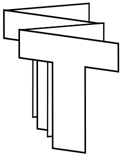
Daniel Kroening, Matt Lewis, Georg Weissenbacher:
Proving Safety with Trace Automata and Bounded Model Checking.
Conference on Formal Methods ’15



```
memset(buf, 0, len);
```



```
void* memset(void *buf, int c, size_t len){
    for(size_t i=0; i<len; i++)
        ((char*)buf)[i]=c;
    return buf;
}
```



```
void* memset(void *buf, int c, size_t len){
    for(size_t i=0; i<len; i++)
        ((char*)buf)[i]=c;
    return buf;
}
```

- `size_t i`: $0 \leq$ `i` $\leq$ `INT_MAX`
- but “standard” acceleration assumes `i` $\in \mathbb{N}$!

- `size_t i`: $0 \le$ `i` $\le$ `INT_MAX`
- but “standard” acceleration assumes `i` $\in \mathbb{N}$!

`i = i +` n for $n >$ (`INT_MAX` $-$ `i`):

(arithmetic overflow)

- `size_t i`: $0 \le \texttt{i} \le \texttt{INT\_MAX}$
- but “standard” acceleration assumes $\texttt{i} \in \mathbb{N}$!

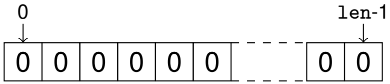
$$\texttt{i = i + } n \text{ for } n > (\texttt{INT\_MAX} - \texttt{i}):$$

(arithmetic overflow)

- *Off-the-shelf* acceleration can
  - miss bugs
  - result in false positives

- *Off-the-shelf* acceleration does not support arrays

- *Off-the-shelf* acceleration does not support arrays
- but content of `buf` matters in `memset(buf, 0, len)`:



Acceleration for Bit-vectors & Arrays [FMSD'15]

- We support bit-vectors

$$\exists n \leq (\texttt{INT\_MAX} - \texttt{i}) \,.\, \texttt{i}' = \texttt{i} + n$$

- as well as arrays

$$\begin{pmatrix} \forall j \leq n \,.\, \texttt{buf}'[\texttt{i} + j] = c \quad \wedge \\ \forall j > n \,.\, \texttt{buf}'[\texttt{i} + j] = \texttt{buf}[\texttt{i} + j] \end{pmatrix}$$

Acceleration for Bit-vectors & Arrays [FMSD'15]

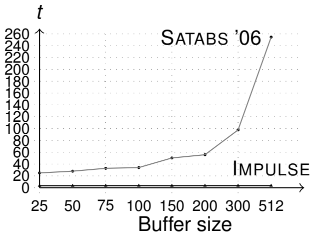
- We support bit-vectors

$$\exists n \leq (\texttt{INT\_MAX} - \texttt{i}) \,.\, \texttt{i}' = \texttt{i} + n$$

- as well as arrays

$$\begin{pmatrix} \forall j \leq n \,.\, \texttt{buf}'[\texttt{i} + j] = c \quad \wedge \\ \forall j > n \,.\, \texttt{buf}'[\texttt{i} + j] = \texttt{buf}[\texttt{i} + j] \end{pmatrix}$$

- Detection of deep bugs (e.g., buffer-overflows) in C programs
  - on *real* GNU systems programs (e.g., Aeon web-server)
  - runtime does *not* depend on number of loop iterations



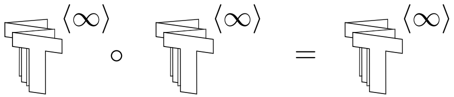
Acceleration for Proving Correctness [FM’15]

- BMC checks whether “no more steps” feasible
- Clashes with acceleration; there are always additional steps:

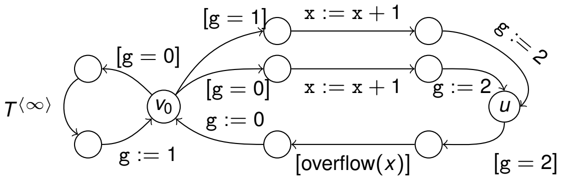
$$T^{\langle\infty\rangle} \circ T^{\langle\infty\rangle} = T^{\langle\infty\rangle}$$

Acceleration for Proving Correctness [FM'15]

- BMC checks whether "no more steps" feasible
- Clashes with acceleration; there are always additional steps:



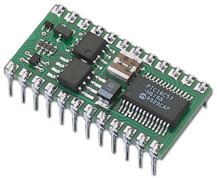
- we use automata to eliminate "redundant" acceleration steps

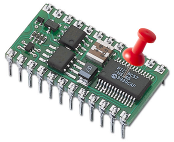


- "Look ma, no fixpoints!"

Hardware

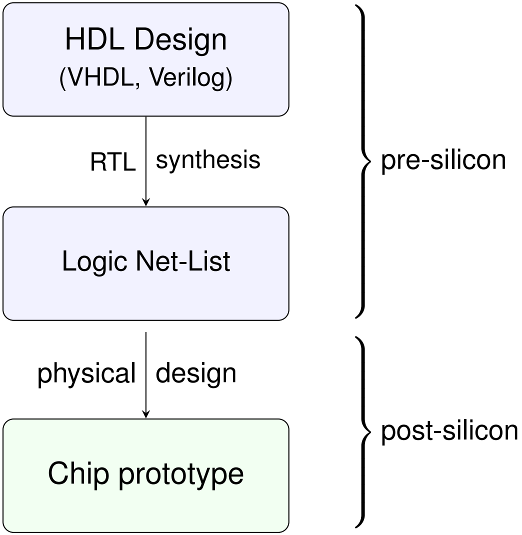
(Integrated Circuits)



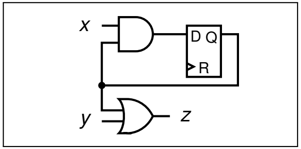


Fault Localization in Post-Silicon

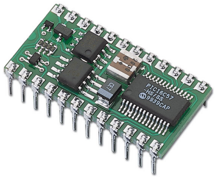
Zhu, Weissenbacher, Malik:
Silicon fault diagnosis using sequence interpolation with backbones.
International Conference on Computer-Aided Design ’14


HDL Design
(VHDL, Verilog)
RTL
synthesis
Logic Net-List
physical
design
Chip prototype
pre-silicon
post-silicon

```
always@(posedge clk)
  if (ue[1]) begin
    IP = IP + len;
    if (btaken)
      IP = IP + dist;
  end
```

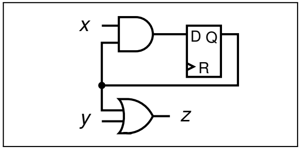


pre-silicon



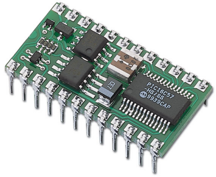
post-silicon

```
always@(posedge clk)
  if (ue[1]) begin
    IP = IP + len;
    if (btaken)
      IP = IP + dist;
  end
```



hardware model checking

[Proc. IEEE'15]

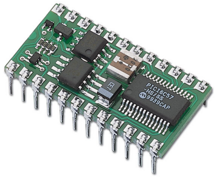


testing

Verified “Golden” Hardware Model

(transition relation T)

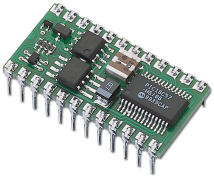
vs.



(silicon prototype)

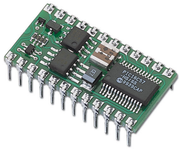
Electrical Faults

Manufacturing process can introduce



- stuck-at faults
- bridging faults
- transistor faults
- ...

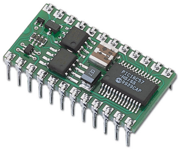
Post-Silicon Fault Localization with Interpolants [ICCAD'14]



crashes in state *f*

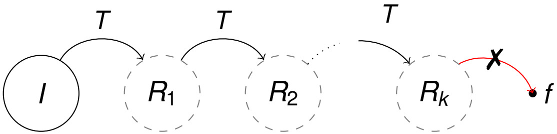
but T does not reflect electrical faults

Post-Silicon Fault Localization with Interpolants [ICCAD'14]



crashes in state f

but T does not reflect electrical faults



Post-Silicon Fault Localization with Interpolants [ICCAD'14]

Verification task:

- Which *gate* in which execution *cycle* causes the discrepancy?

Challenge:

- On-chip at-speed executions can be extremely long
- States in integrated circuit not fully observable

Post-Silicon Fault Localization with Interpolants [ICCAD’14]

Verification task:

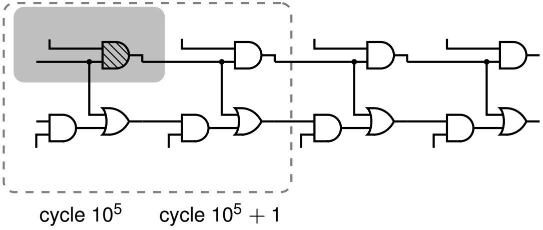
- Which *gate* in which execution *cycle* causes the discrepancy?

Challenge:

- On-chip at-speed executions can be extremely long
- States in integrated circuit not fully observable

Solution:

- Use *interpolation* to analyze *windows of cycles* individually



Post-Silicon Fault Localization with Interpolants [ICCAD’14]

Verification task:

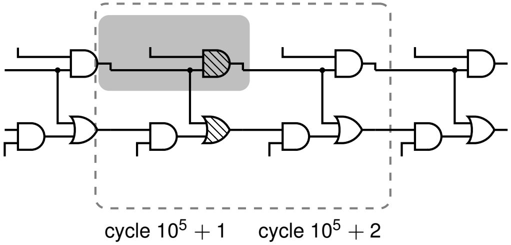
- Which *gate* in which execution *cycle* causes the discrepancy?

Challenge:

- On-chip at-speed executions can be extremely long
- States in integrated circuit not fully observable

Solution:

- Use *interpolation* to analyze *windows of cycles* individually



Post-Silicon Fault Localization with Interpolants [ICCAD'14]

Verification task:

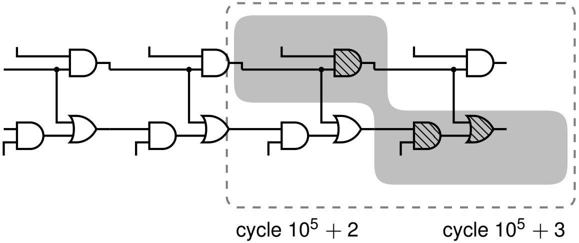
- Which *gate* in which execution *cycle* causes the discrepancy?

Challenge:

- On-chip at-speed executions can be extremely long
- States in integrated circuit not fully observable

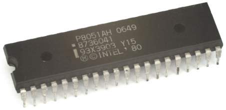
Solution:

- Use *interpolation* to analyze *windows of cycles* individually



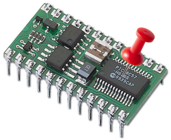
Post-Silicon Fault Localization with Interpolants [ICCAD’14]

- Scalable fault diagnosis for post-silicon
- Evaluated on micro-controller designs 68HC05 and 8051



Scalable Software Model Checking [CAV'14]

Efficient Detection of "Deep" Bugs [FMSD'15] (CAV'13), [FM'15]



Fault Localization in Post-Silicon [ICCAD'14]

Thank You

Logical foundations [JAR'16] (single auth. SAT'12)

State-of-the-Art [Proc. IEEE'15]